Polyraptor: Embracing Path and Data Redundancy in Data Centres for Efficient Data Transport

Mohammed Alasmar
Department of Informatics
University of Sussex
M.Alasmar@sussex.ac.uk

George Parisis
Department of Informatics
University of Sussex
G.Parisis@sussex.ac.uk

Jon Crowcroft
Computer Laboratory
University of Cambridge
Jon.Crowcroft@cl.cam.ac.uk

ABSTRACT

In this paper, we introduce Polyraptor, a novel data transport protocol that uses RaptorQ (RQ) codes and is tailored for one-to-many and many-to-one data transfer patterns, which are extremely common in modern data centres. Polyraptor builds on previous work on fountain coding-based transport and provides excellent performance, by exploiting native support for multicasting in data centres and data resilience provided by data replication.

1 INTRODUCTION

Data centres support the provision of core Internet services, such as search, social networking, cloud computing and video streaming. Data Centre Networks (DCNs) consist of a large number of commodity servers and switches, support multiple paths among servers and very large aggregate bandwidth. TCP is ill-suited for meeting the throughput and latency requirements of applications in DCNs. Exploiting multiple paths and maximising resources' utilisation, while network congestion is fairly dealt with, has therefore been a prominent research area [1][4].

A range of application workloads in modern data centres involve one-to-many and many-to-one traffic exchange. For example, distributed storage systems, such as GFS [6], replicate data blocks, but clients are constrained by the underlying unicast transport protocol when storing data to multiple servers (one-to-many) and fetching data that is available on multiple servers (many-to-one). Partition-aggregate application workloads are similarly constrained, as they make use of underlying distributed storage systems.

Polyraptor builds on our previous work [2] and is tailored for one-to-many and many-to-one data transfer patterns, supports multi-path transport, eliminates Incast and can work well with shallow buffers in network switches (§ 2). Polyraptor uses RQ codes [3] and follows a receiver-driven approach for flow and congestion control, which is reminiscent to NDP [4]. We have implemented a simulation model of Polyraptor [1] and compared its performance to standard unicast data transport (§ 3).

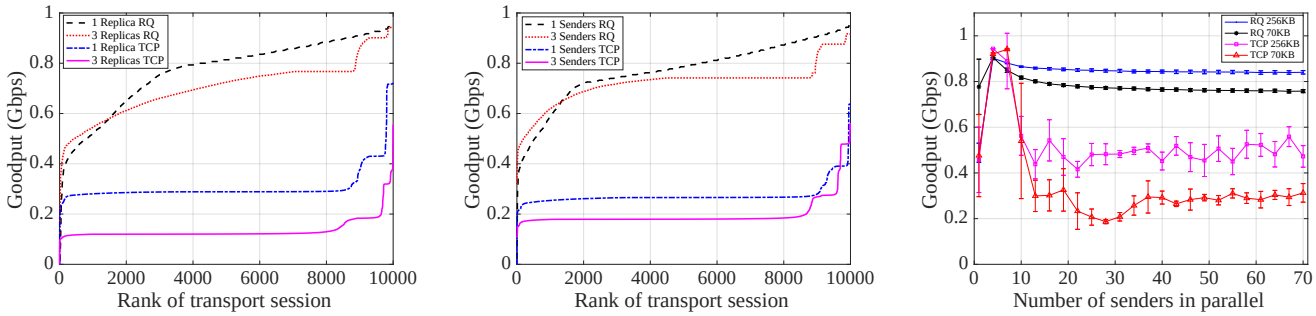
2 DESIGN

Polyraptor employs a receiver-driven communication model, where receivers actively manage the rate at which encoding symbols arrive (effectively providing flow and congestion control), by explicitly requesting symbols from senders. RQ codes are rateless and systematic; encoding symbols consist of the source symbols (i.e. original data fragments), along with a potentially very large number of repair symbols. In Polyraptor, source symbols are sent at the beginning of a session, followed by repair symbols, as required by receivers. In the absence of loss, source symbols are immediately passed to the application without inducing any penalty in terms of decoding latency; this is particularly desirable for short flows that are commonly latency-sensitive. RQ codes have excellent performance in terms of network overhead, decoding latency and failure probability[3][2].

Polyraptor sessions. A Polyraptor session may involve one sender and multiple receivers or multiple senders and one receiver (unicast data transport is a specialisation of one of the above scenarios). A sender first sends a whole window of encoded symbols at line rate for the first RTT; receivers then take control of the data transfer by requesting encoded symbols (by sending pull requests). We adopt NDP's switching architecture [4], which supports two different packet queues: a priority header queue and a data queue. When the data queue overflows, incoming encoding symbols are trimmed and the resulting headers get priority forwarding. The data transport layer at each receiver has only one pull queue shared by all sessions. A pull request is added to this queue upon receiving a full or trimmed symbol. The receiver then paces pull packets across all sessions, so that the aggregate data rate matches the receiver's link capacity. These pull packets trigger the sending of new encoded symbols. A lost symbol does not have to be re-requested. Instead, a new symbol will contribute to the decoding process equally to the lost one. This, along with symbol trimming, is crucial for supporting an Incast-free protocol. Packet loss and out-of-order packets don't hurt performance as they do in TCP, thus there is no need to extensively buffer packets to minimise

[1] We have implemented Polyraptor as an OMNet++ model and made the source code available at: https://github.com/mzsala/polyraptor.

[2] Decoding fails only 1 in 1,000,000 when the receiver collects $n + 2$ encoding symbols, n being the number of original fragments [3].



(a) Multicast **(b) Multi-source** **(c) Incast**

Figure 1: Goodput results at a 250 servers FatTree topology (1GB link speed & 10μs link delay). 20% of the sessions are background traffic. The presented results are for the rest 80% of the sessions (4 MB each) with arrival times follow a Poisson process with $\lambda = 2560$. Session (flow) scheduling follows a permutation traffic matrix. Error bars in (c) represent 95% confidence interval. Each experiment is the average of 5 repetitions using different seeds.

losses; symbols can be sprayed in the network, exploiting all available (equal-cost) paths.

Multi-source transport. In Polyraptor, a receiver can pull encoding symbols from multiple senders. RQ symbols are equally useful for decoding the original data, if no duplicate symbols are received. This can be achieved without any coordination; senders independently seed the underlying pseudorandom generator that is used to encode symbols [3], therefore collectively producing statistically unique symbols. Senders initially select and send a subset of source symbols (exploiting the systematic nature of RQ codes), before sending statistically unique repair symbols. The number of senders is known at session establishment, therefore a simple partitioning of the source symbols would ensure absence of duplicate symbols at the receiver side. Multi-source transport enables a natural load balancing mechanism where each server contributes symbols at its available capacity.

Multicast transport. The rateless and systematic nature of RQ codes makes them ideal for multicasting data. A sender initially pushes a window of encoding symbols to all receivers, which then start pulling additional (source and repair) ones. A Polyraptor sender aggregates pull requests and multicasts a new symbol only after all receivers have sent one. As part of our current work is to be able to detect and eliminate straggler receivers by detaching them from the group and exchanging symbols with them independently through a one-to-one Polyraptor session.

3 DISCUSSION

In Figure 1a, we present Polyraptor's performance in a distributed storage scenario with 1 and 3 replicas. The three replica servers are randomly selected outside the client's rack. We have emulated the same behaviour with TCP by multi-unicasting data to the randomly selected servers. We have simulated 10,000 sessions (flows). Our multicasting model follows the design in [5]. Polyraptor maintains excellent performance when replicating data to 3 servers due to the underlying multicast support. Packet trimming along with RQ coding provide resilience against transient and persistent congestion. In order to demonstrate Polyraptor's performance when multi-sourcing data, we simulated a distributed storage scenario where a client fetches data from 1 and 3 replica servers at the same time. We emulated this behaviour with TCP by assuming that storage servers transfer back to the client part of the requested blocks without requiring any coordination. Figure 1b follows the same pattern as Figure 1a. Polyraptor sustains excellent performance fully utilising all available data replicas and the underlying network resources. Figure 1c presents a classic Incast scenario with synchronised short flows. Packet trimming along with the rateless nature of the RQ codes result in Incast elimination.

The presented work is not conclusive of the full potential of Polyraptor. As part of our current work, we are evaluating Polyraptor's behaviour under different workloads and the existence of network hotspots. We are also looking at the influence of RQ encoding/decoding complexity and latency as well as decoding failure probability in the performance of Polyraptor for different application workloads and resulting size of the blocks that are being encoded/decoded.

REFERENCES

[1] C. Raiciu et al. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proc. of SIGCOMM.*

[2] G.Parisis et al. 2013. Trevi: Watering Down Storage Hotspots with Cool Fountain Codes. In *Proc. of HotNets.*

[3] M.Luby et al. [n. d.]. RaptorQ Forward Error Correction Scheme for Object Delivery. *IETF, RFC 6330, 2011* ([n. d.]).

[4] M. Handley et al. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proc. of SIGCOMM.*

[5] N.Mohammad et al. 2017. DCCast: Efficient Point to Multipoint Transfers Across Datacenters. In *HotCloud Workshop.* USENIX.

[6] S. Ghemawat et. al. 2003. The Google File System. In *SOSP.*